CS 267 Applications of Parallel Computers

Lecture 14:

Graph Partitioning - II

Bob Lucas

derived from earlier lectures by Jim Demmel and Dave Culler

www.nersc.gov/~dhbailey/cs267

Outline of Graph Partitioning Lectures

° **Review of last lecture**

° **Partitioning without Nodal Coordinates - continued**

- **Kernighan/Lin**
- **Spectral Partitioning**

° **Multilevel Acceleration**

- **BIG IDEA, will appear often in course**

° **Available Software**

- **good sequential and parallel software availble**

° **Comparison of Methods**

Review Definition of Graph Partitioning

° Given a graph $G = (N, E, W_N, W_E)$

- N = nodes (or vertices), E = edges
- W_N = node weights, W_E = edge weights

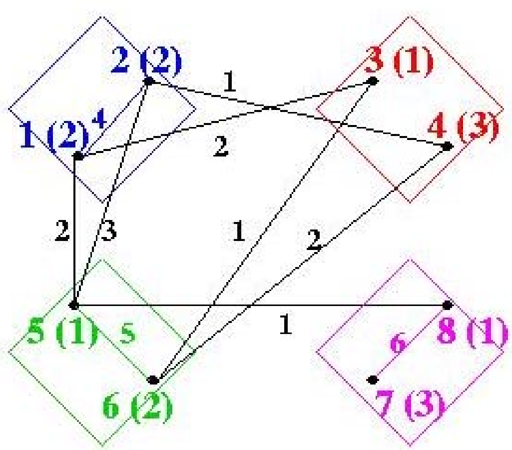
° Ex: N = {tasks}, W_N = {task costs}, edge (j,k) in E means task j sends $W_E(j,k)$ words to task k

° Choose a partition $N = N_1 \cup N_2 \cup \ldots \cup N_P$ such that

- The sum of the node weights in each N_j is "about the same"
- The sum of all edge weights of edges connecting all different pairs N_j and N_k is minimized

° Ex: balance the work load, while minimizing communication

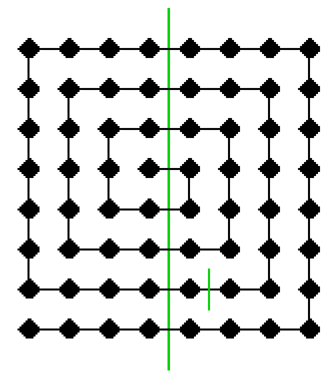
° Special case of $N = N_1 \cup N_2$: Graph Bisection



Review of last lecture

° **Partitioning with nodal coordinates**

- **Rely on graphs having nodes connected (mostly) to "nearest neighbors" in space**
- **Common when graph arises from physical model**
- **Algorithm very efficient, does not depend on edges!**
- **Can be used as good starting guess for subsequent partitioners, which do examine edges**
- **Can do poorly if graph less connected:**



° **Partitioning without nodal coordinates**

- **Depends on edges**
- **No assumptions about where "nearest neighbors" are**
- **Began with Breadth First Search (BFS)**

Partitioning without nodal coordinates - Kernighan/Lin

° Take a initial partition and iteratively improve it

- Kernighan/Lin (1970), cost = $O(|N|^3)$ but easy to understand
- Fiduccia/Mattheyses (1982), cost = $O(|E|)$, much better, but more complicated

° Let $G = (N,E,W_E)$ be partitioned as $N = A \cup B$, where $|A| = |B|$

° $T = \text{cost}(A,B) = \Sigma$ {W(e) where e connects nodes in A and B}

° Find subsets X of A and Y of B with $|X| = |Y|$ so that swapping X and Y decreases cost:

- newA = A - X U Y and newB = B - Y U X
- newT = cost(newA , newB) < cost(A,B)
- Keep choosing X and Y until cost no longer decreases

° Need to compute newT efficiently for many possible X and Y, choose smallest

Kernighan/Lin Algorithm

```
Compute  T = cost(A,B) for initial A, B                        … cost = O(|N|^2)
Repeat
     … One pass greedily computes |N|/2 possible X,Y to swap, picks best
     Compute costs D(n) for all n in N                          … cost = O(|N|^2)
     Unmark all nodes in N                                      … cost = O(|N|)
     While there are unmarked nodes                             … |N|/2 iterations
         Find an unmarked pair (a,b) maximizing gain(a,b)       … cost = O(|N|^2)
         Mark a and b (but do not swap them)                    … cost = O(1)
         Update D(n) for all unmarked n,
              as though a and b had been swapped                … cost = O(|N|)
     Endwhile
         … At this point we have computed a sequence of pairs
         …  (a1,b1), … , (ak,bk)   and gains gain(1),…., gain(k)
         … where k = |N|/2, numbered in the order in which we marked them
     Pick m maximizing Gain = Σk=1 to m  gain(k)                … cost = O(|N|)
         … Gain is reduction in cost from swapping (a1,b1) through (am,bm)
     If Gain > 0 then   … it is worth swapping
         Update newA = A - { a1,…,am } U { b1,…,bm }            … cost = O(|N|)
         Update newB = B - { b1,…,bm } U { a1,…,am }            … cost = O(|N|)
         Update T = T - Gain                                    … cost = O(1)
     endif
Until Gain <= 0
```

Comments on Kernighan/Lin Algorithm

° **Most expensive line show in red**

° **Some gain(k) may be negative, but if later gains are large, then final Gain may be positive**

- can escape "local minima" where switching no pair helps

° **How many times do we Repeat?**

- K/L tested on very small graphs (|N|<=360) and got convergence after 2-4 sweeps
- For random graphs (of theoretical interest) the probability of convergence in one step appears to drop like $2^{-|N|/30}$

Partitioning without nodal coordinates - Spectral Bisection

° Based on theory of Fiedler (1970s), popularized by Pothen, Simon, Liou (1990)

° Motivation, by analogy to a vibrating string

° Basic definitions

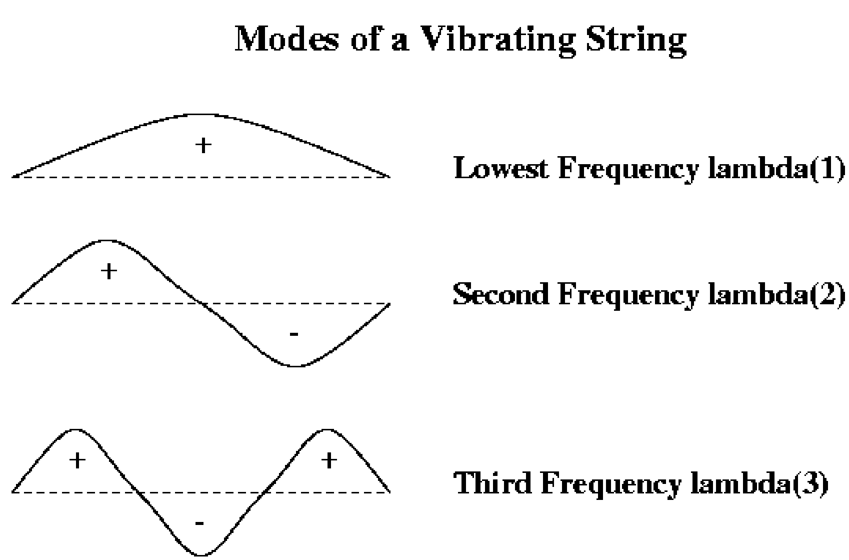
° Vibrating string, revisited

° Implementation via the Lanczos Algorithm

- To optimize sparse-matrix-vector multiply, we graph partition
- To graph partition, we find an eigenvector of a matrix associated with the graph
- To find an eigenvector, we do sparse-matrix vector multiply
- No free lunch ...

Motivation for Spectral Bisection: Vibrating String

- Think of G = 1D mesh as masses (nodes) connected by springs (edges), i.e. a string that can vibrate
- Vibrating string has modes of vibration, or harmonics
- Label nodes by whether mode - or + to partition into N- and N+
- Same idea for other graphs (eg planar graph ~ trampoline)



Basic Definitions

- *Definition*: The incidence matrix In(G) of a graph G(N,E) is an |N| by |E| matrix, with one row for each node and one column for each edge. If edge e=(i,j) then column e of In(G) is zero except for the i-th and j-th entries, which are +1 and -1, respectively.

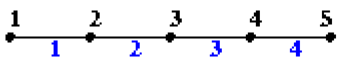
- Slightly ambiguous definition because multiplying column e of In(G) by -1 still satisfies the definition, but this won't matter...

- *Definition*: The Laplacian matrix L(G) of a graph G(N,E) is an |N| by |N| symmetric matrix, with one row and column for each node. It is defined by
  - L(G) (i,i) = degree of node I (number of incident edges)
  - L(G) (i,j) = -1 if i != j and there is an edge (i,j)
  - L(G) (i,j) = 0 otherwise

Example of In(G) and L(G) for 1D and 2D meshes

Incidence and Laplacian Matrices

Graph G

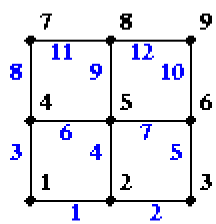


Incidence Matrix In(G)

	1	2	3	4
1	-1			
2	1	-1		
3		1	-1	
4			1	-1
5				1

Laplacian Matrix L(G)

	1	2	3	4	5
1	1	-1			
2	-1	2	-1		
3		-1	2	-1	
4			-1	2	-1
5				-1	1



	1	2	3	4	5	6	7	8	9	10	11	12
1	-1		1									
2	1	-1		1								
3		1			1							
4			-1			-1		1				
5				-1		1	-1		1			
6					-1		1			1		
7								-1			-1	
8									-1		1	-1
9										-1		1

	1	2	3	4	5	6	7	8	9
1	2	-1		-1					
2	-1	3	-1		-1				
3		-1	2			-1			
4	-1			3	-1		-1		
5		-1		-1	4	-1		-1	
6			-1		-1	3			-1
7				-1			2	-1	
8					-1		-1	3	-1
9						-1		-1	2

Nodes numbered in black

Edges numbered in blue

Properties of Incidence and Laplacian matrices

° *Theorem 1:* Given G, In(G) and L(G) have the following properties (proof on web page)

- L(G) is symmetric. (This means the eigenvalues of L(G) are real and its eigenvectors are real and orthogonal.)
- Let $e = [1,\ldots,1]^T$, i.e. the column vector of all ones. Then L(G)*e=0.
- $In(G) * (In(G))^T = L(G)$. This is independent of the signs chosen for each column of In(G).
- Suppose $L(G)*v = \lambda*v$, $v \neq 0$, so that v is an eigenvector and λ an eigenvalue of L(G). Then

$$\lambda = \| In(G)^T * v \|^2 / \| v \|^2 \qquad \ldots \|x\|^2 = \Sigma_k x_k^2$$
$$= \Sigma \{ (v(i)-v(j))^2 \text{ for all edges } e=(i,j) \} / \Sigma_i v(i)^2$$

- The eigenvalues of L(G) are nonnegative:
  - $0 = \lambda_1 <= \lambda_2 <= \ldots <= \lambda_n$
- The number of connected components of G is equal to the number of λ_i equal to 0. In particular, $\lambda_2 != 0$ if and only if G is connected.

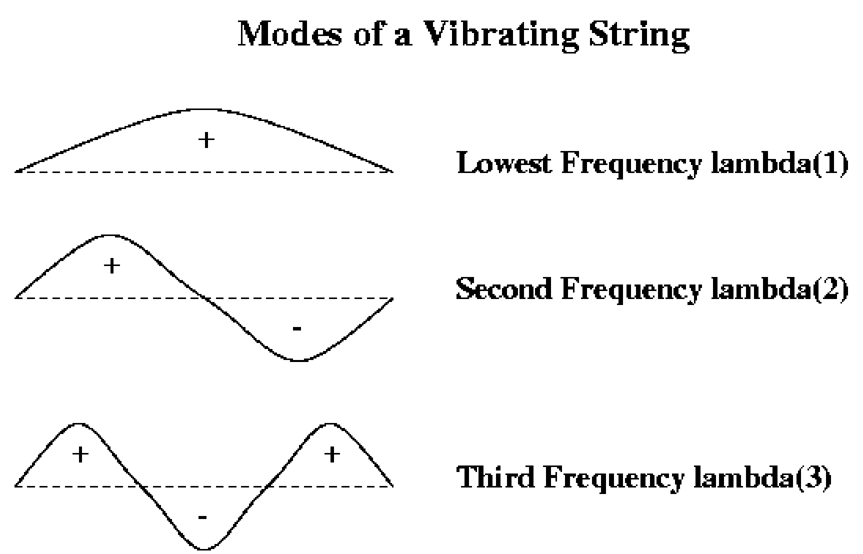
° *Definition*: $\lambda_2(L(G))$ is the algebraic connectivity of G

Spectral Bisection Algorithm

- **Spectral Bisection Algorithm:**
  - Compute eigenvector v_2 corresponding to $\lambda_2(L(G))$
  - For each node n of G
    - if $v_2(n) < 0$ put node n in partition N-
    - else put node n in partition N+
- **Why does this make sense? First reasons...**
- ***Theorem 2 (Fiedler, 1975):*** **Let G be connected, and N- and N+ defined as above. Then N- is connected. If no $v_2(n) = 0$, then N+ is also connected.** (proof on web page)
- **Recall $\lambda_2(L(G))$ is the algebraic connectivity of G**
- ***Theorem 3 (Fiedler):*** **Let $G_1(N,E_1)$ be a subgraph of G(N,E), so that G_1 is "less connected" than G. Then $\lambda_2(L(G)) <= \lambda_2(L(G))$, i.e. the algebraic connectivity of G_1 is less than or equal to the algebraic connectivity of G.** (proof on web page)

Motivation for Spectral Bisection: Vibrating String

- Vibrating string has modes of vibration, or harmonics
- Modes computable as follows
  - Model string as masses connected by springs (a 1D mesh)
  - Write down F=ma for coupled system, get matrix A
  - Eigenvalues and eigenvectors of A are frequencies and shapes of modes
- Label nodes by whether mode - or + to get N- and N+
- Same idea for other graphs (eg planar graph ~ trampoline)



Details for vibrating string

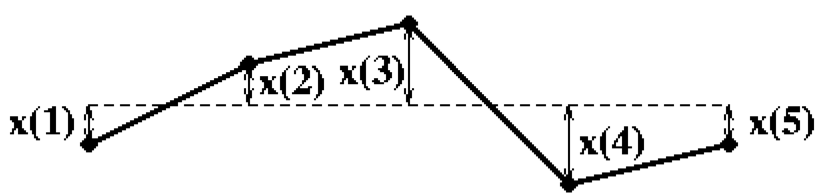
- Force on mass j = k*[x(j-1) - x(j)] + k*[x(j+1) - x(j)]

  = -k*[-x(j-1) + 2*x(j) - x(j+1)]
- F=ma yields m*x''(j) = -k*[-x(j-1) + 2*x(j) - x(j+1)] (*)
- Writing (*) for j=1,2,…,n yields

$$m * \frac{d^2}{dx^2}\begin{pmatrix} x(1) \\ x(2) \\ \dots \\ x(j) \\ \dots \\ x(n) \end{pmatrix} = -k*\begin{pmatrix} 2*x(1) - x(2) \\ -x(1) + 2*x(2) - x(3) \\ \dots \\ -x(j-1) + 2*x(j) - x(j+1) \\ \dots \\ 2*x(n-1) - x(n) \end{pmatrix} = -k*\begin{pmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & & \dots & & & \\ & & -1 & 2 & -1 & \\ & & & & \dots & \\ & & & & -1 & 2 \end{pmatrix} * \begin{pmatrix} x(1) \\ x(2) \\ \dots \\ x(j) \\ \dots \\ x(n) \end{pmatrix} = -k*L*\begin{pmatrix} x(1) \\ x(2) \\ \dots \\ x(j) \\ \dots \\ x(n) \end{pmatrix}$$

$$(-m/k)\, x'' = L*x$$

Vibrating Mass Spring System



Details for vibrating string - continued

° $-(m/k)\, x'' = L^*x$, where $x = [x_1, x_2, \ldots, x_n]^T$

° Seek solution of form $x(t) = \sin(\alpha^* t) * x0$

- $L^*x0 = (m/k)^*\alpha^2 * x0 = \lambda * x0$
- For each integer i, get $\lambda = 2^*(1-\cos(i^*\pi/(n+1))$, $x0 = \begin{pmatrix} \sin(1^*i^*\pi/(n+1)) \\ \sin(2^*i^*\pi/(n+1)) \\ \ldots \\ \sin(n^*i^*\pi/(n+1)) \end{pmatrix}$
- Thus x0 is a sine curve with frequency proportional to i
- Thus $\alpha^2 = 2^*k/m\ ^*(1-\cos(i^*\pi/(n+1))$ or $\alpha \sim \mathrm{sqrt}(k/m)^*\pi^*i/(n+1)$

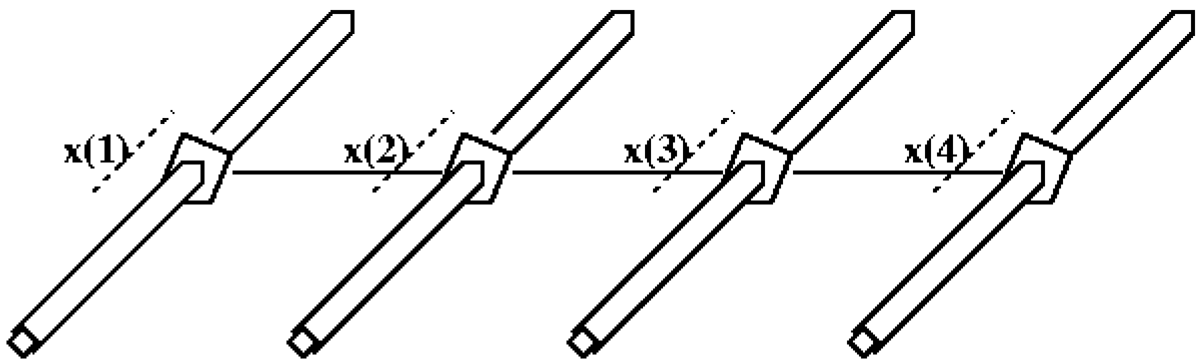
° $L = \begin{pmatrix} 2 & -1 & & \\ -1 & 2 & -1 & \\ & \cdots & & \\ & & -1 & 2 \end{pmatrix}$ not quite L(1D mesh),

but we can fix that ...

A "vibrating string" for L(1D mesh)

- First equation changes to $m*x''(1) = -k*[-x(2)+ \cancel{2}x(1)]$
    - First row of T changes from [2 -1 0 …] to [1 -1 0 …]
- Last equation changes to $m*x''(n)=-k*[-x(n-1) + \cancel{2}x(n)]$
    - Last row of T changes from [… 0 -1 2] to [… 0 -1 1]
- Component j of i-th eigenvector changes to $\cos((j-.5)*(i-1)*\pi/n)$

"Vibrating String" for Spectral Bisection

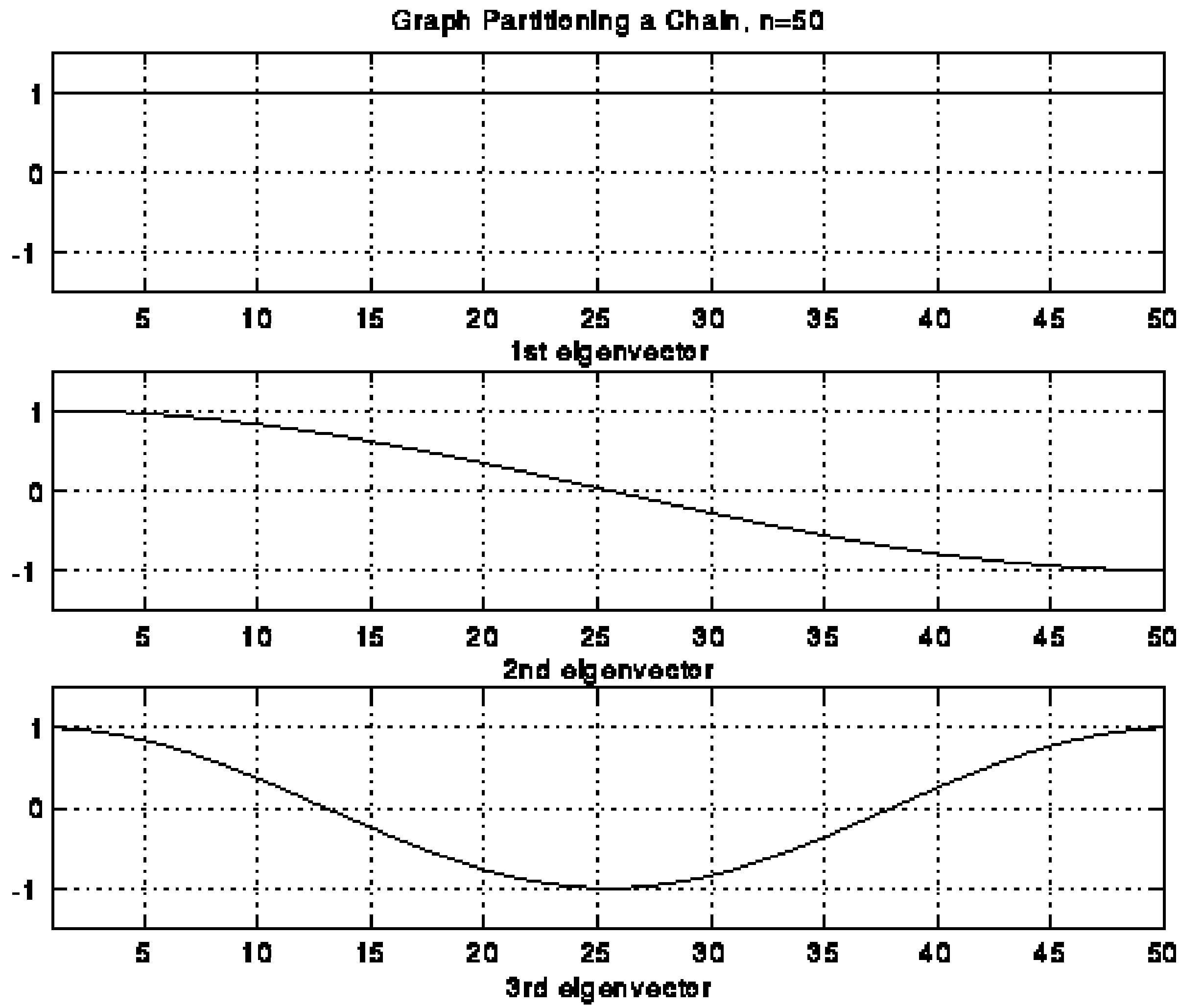


Eigenvectors of L(1D mesh)

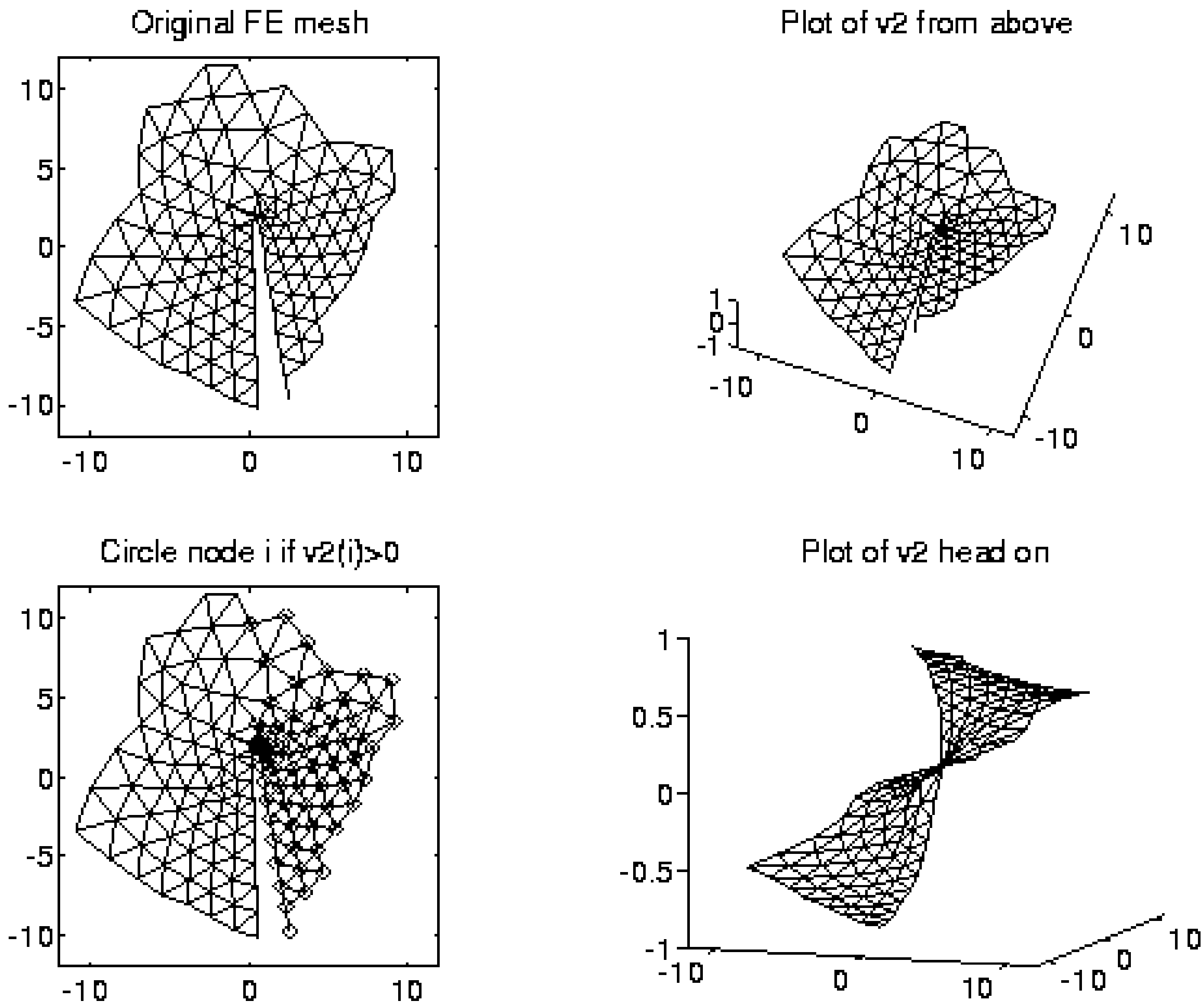
Eigenvector 1
(all ones)

Eigenvector 2

Eigenvector 3

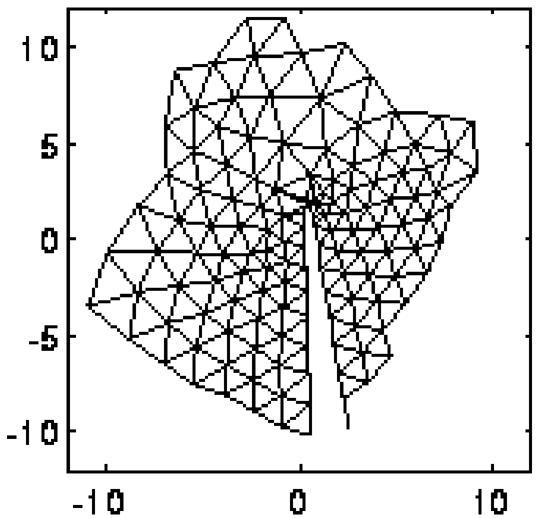


2nd eigenvector of L(planar mesh)

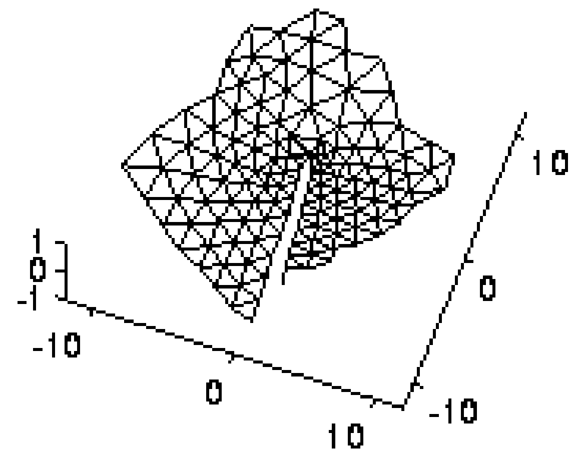


4th eigenvector of L(planar mesh)

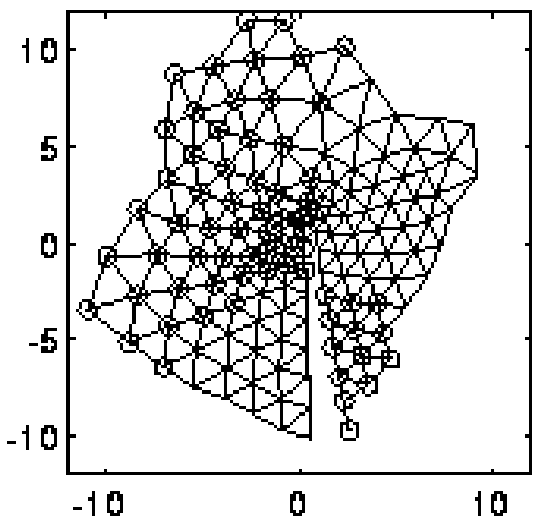
Original FE mesh



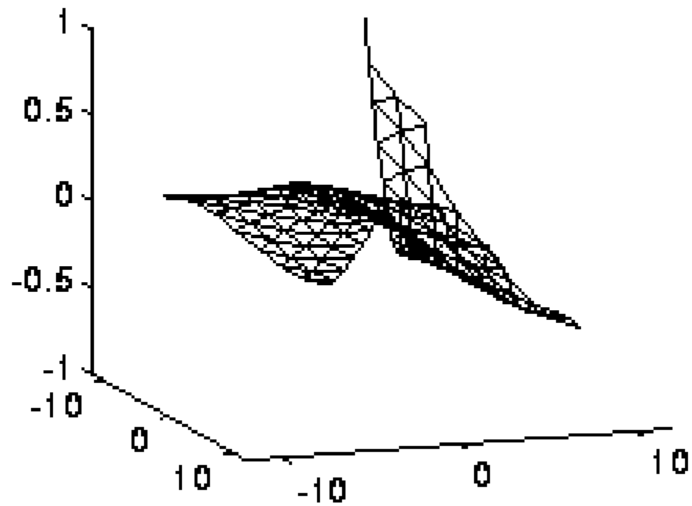
Plot of v4 from above



Circle node i if v4(i)>0



Plot of v4 head on



Computing v_2 and λ_2 of L(G) using Lanczos

- Given any n-by-n symmetric matrix A (such as L(G)) Lanczos computes a k-by-k "approximation" T by doing k matrix-vector products, k << n

```
Choose an arbitrary starting vector r
b(0) = ||r||
j=0
repeat
    j=j+1
    q(j) = r/b(j-1)             … scale a vector
    r = A*q(j)                  … matrix vector multiplication, the most expensive step
    r = r - b(j-1)*v(j-1)       … "saxpy", or scalar*vector + vector
    a(j) = v(j)^T * r           … dot product
    r = r - a(j)*v(j)           … "saxpy"
    b(j) = ||r||                … compute vector norm
until convergence               … details omitted
```

$$T = \begin{pmatrix} a(1) & b(1) & & & & \\ b(1) & a(2) & b(2) & & & \bigcirc \\ & b(2) & a(3) & b(3) & & \\ & & \dots & \dots & \dots & \\ \bigcirc & & & b(k-2) & a(k-1) & b(k-1) \\ & & & & b(k-1) & a(k) \end{pmatrix}$$

- Approximate A's eigenvalues/vectors using T's

References

- Details of all proofs on web page
- A. Pothen, H. Simon, K.-P. Liou, “Partitioning sparse matrices with eigenvectors of graphs”, SIAM J. Mat. Anal. Appl. 11:430-452 (1990)
- M. Fiedler, “Algebraic Connectivity of Graphs”, Czech. Math. J., 23:298-305 (1973)
- M. Fiedler, Czech. Math. J., 25:619-637 (1975)
- B. Parlett, “The Symmetric Eigenproblem”, Prentice-Hall, 1980
- www.cs.berkeley.edu/~ruhe/lantplht/lantplht.html
- www.netlib.org/laso

Introduction to Multilevel Partitioning

° **If we want to partition G(N,E), but it is too big to do efficiently, what can we do?**

- **1) Replace G(N,E) by a coarse approximation $G_C(N_C,E_C)$, and partition G_C instead**
- **2) Use partition of G_C to get a rough partitioning of G, and then iteratively improve it**

° **What if G_C still too big?**

- **Apply same idea recursively**

Multilevel Partitioning - High Level Algorithm

(N+,N-) = Multilevel_Partition(N, E)

… recursive partitioning routine returns N+ and N- where N = N+ U N-

if |N| is small

(1) Partition G = (N,E) directly to get N = N+ U N-

Return (N+, N-)

else

(2) Coarsen G to get an approximation $G_C = (N_C, E_C)$

(3) (N_C+ , N_C-) = Multilevel_Partition(N_C, E_C)

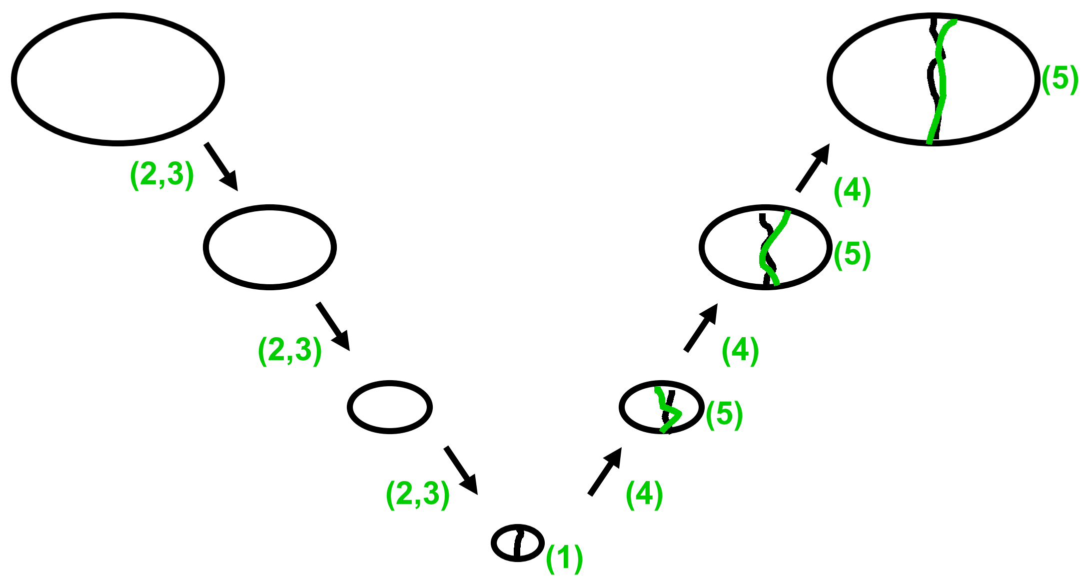
(4) Expand (N_C+ , N_C-) to a partition (N+ , N-) of N

(5) Improve the partition (N+ , N-)

Return (N+ , N-)

endif

“V - cycle:”



How do we

Coarsen?

Expand?

Improve?

Multilevel Kernighan-Lin

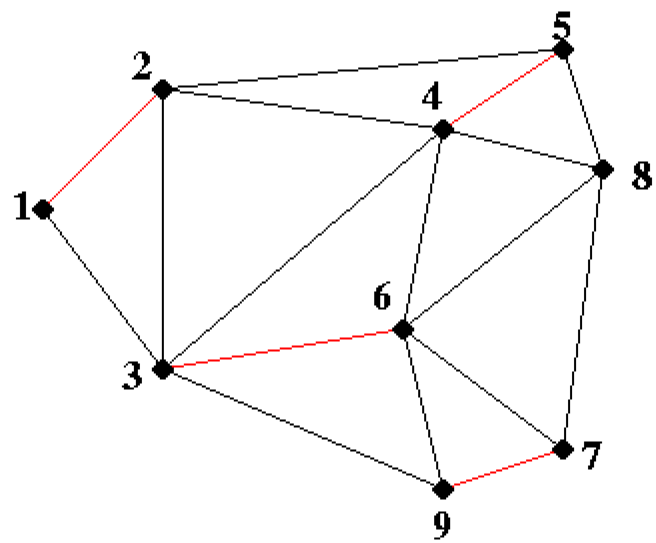
- Coarsen graph and expand partition using maximal matchings
- Improve partition using Kernighan-Lin

Maximal Matching

- *Definition*: A matching of a graph G(N,E) is a subset E_m of E such that no two edges in E_m share an endpoint
- *Definition:* A maximal matching of a graph G(N,E) is a matching E_m to which no more edges can be added and remain a matching
- A simple greedy algorithm computes a maximal matching:

```
let E_m be empty
mark all nodes in N as unmatched
for i = 1 to |N|      … visit the nodes in any order
    if i has not been matched
          if there is an edge e=(i,j)  where j is also unmatched,
                add e to E_m
                mark i and j as matched
          endif
    endif
endfor
```

Maximal Matching - Example



Coarsening using a maximal matching

Construct a maximal matching E_m of G(N,E)

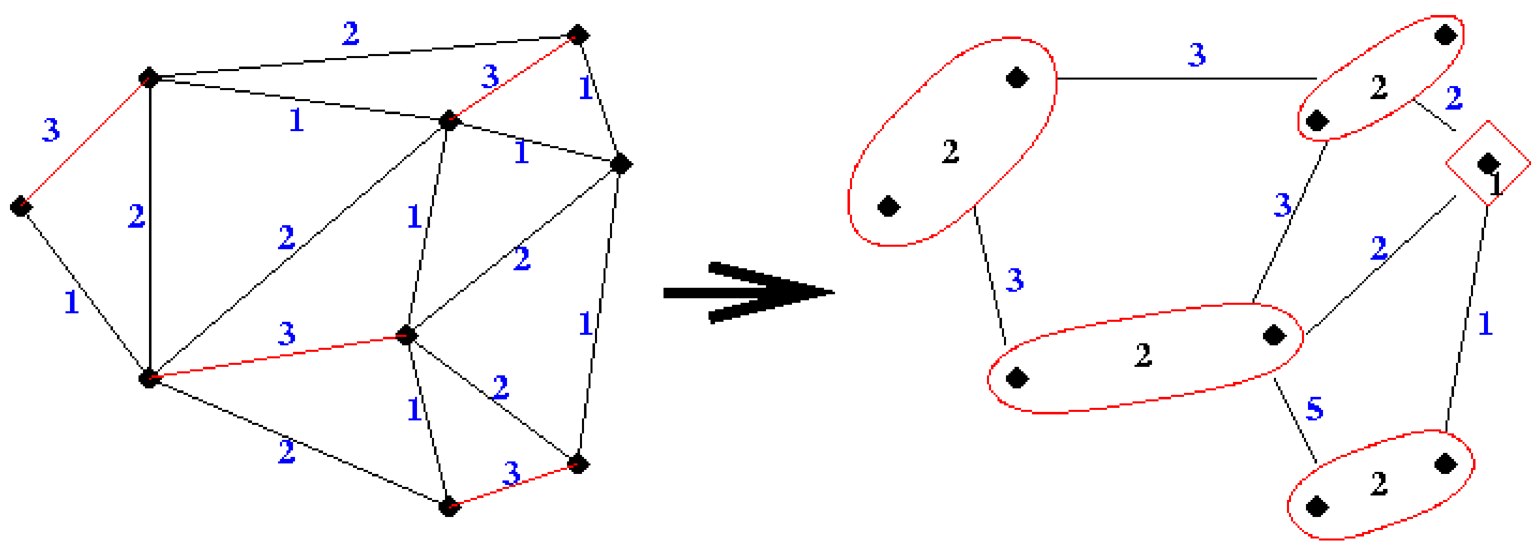
for all edges e=(j,k) in E_m
    Put node n(e) in N_c
    W(n(e)) = W(j) + W(k)   … gray statements update node/edge weights
for all nodes n in N not incident on an edge in E_m
    Put n in N_c … do not change W(n)
… Now each node r in N is "inside" a unique node n(r) in N_c

… Connect two nodes in Nc if nodes inside them are connected in E
for all edges e=(j,k) in E_m
    for each other edge e'=(j,r) in E incident on j
        Put edge ee = (n(e),n(r)) in E_c
        W(ee) = W(e')
    for each other edge e'=(r,k) in E incident on k
        Put edge ee = (n(r),n(e)) in E_c
        W(ee) = W(e')

If there are multiple edges connecting two nodes in N_c, collapse them, adding edge weights

Example of Coarsening

How to coarsen a graph using a maximal matching



$G = (N, E)$

E_m is shown in red

Edge weights shown in blue

Node weights are all one

$G_c = (N_c, E_c)$

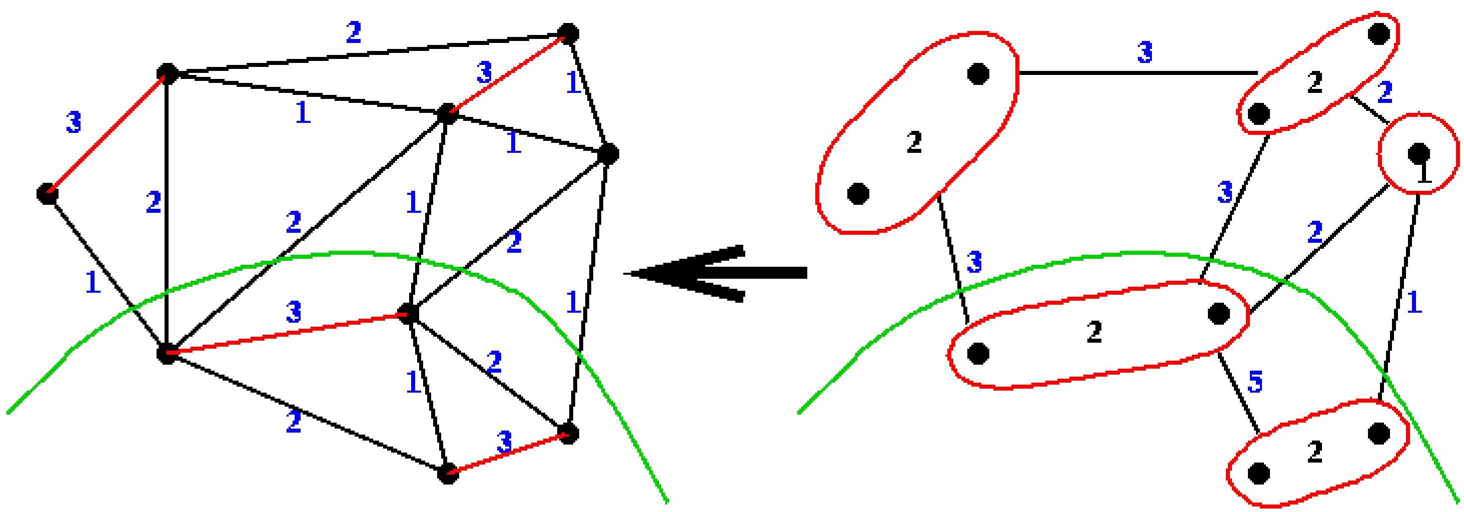
N_c is shown in red

Edge weights shown in blue

Node weights shown in black

Expanding a partition of G_C to a partition of G

Converting a coarse partition to a fine partition



Partition shown in green

Multilevel Spectral Bisection

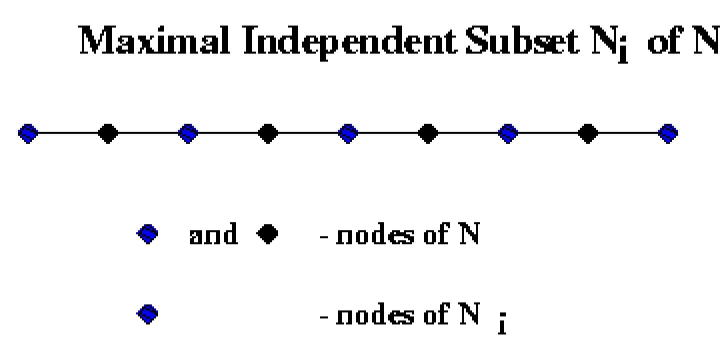
° Coarsen graph and expand partition using maximal independent sets

° Improve partition using Rayleigh Quotient Iteration

Maximal Independent Sets

- *Definition*: An independent set of a graph $G(N,E)$ is a subset N_i of N such that no two nodes in N_i are connected by an edge
- *Definition:* A maximal independent set of a graph $G(N,E)$ is an independent set N_i to which no more nodes can be added and remain an independent set
- A simple greedy algorithm computes a maximal independent set:

```
let Ni be empty
for i = 1 to |N|      … visit the nodes in any order
    if  node i is not adjacent to any node already in Ni
        add i to Ni
    endif
endfor
```

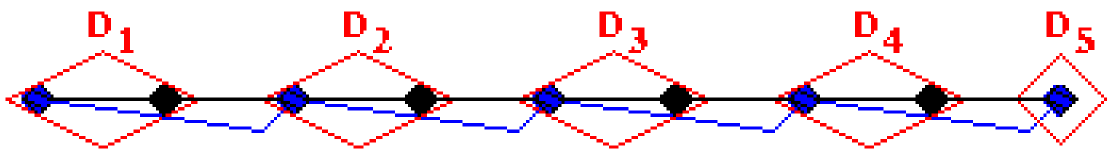


Coarsening using Maximal Independent Sets

```
… Build "domains" D(i) around each node i in N_i to get nodes in N_c
… Add an edge to E_c whenever it would connect two such domains
E_c = empty set
for all nodes i in N_i
    D(i) = ( {i}, empty set )
    … first set contains nodes in D(i), second set contains edges in D(i)
unmark all edges in E
repeat
    choose an unmarked edge e = (i,j) from E
    if exactly one of i and j (say i) is in some D(k)
        mark e
        add j and e to D(k)
    else if i and j are in two different D(k)'s (say D(ki) and D(kj))
        mark e
        add edge (ki, kj) to E_c
    else if both i and j are in the same D(k)
        mark e
        add e to D(k)
    else
        leave e unmarked
    endif
until no unmarked edges
```

Example of Coarsening

Computing G_c from G



and - nodes of N

- nodes of N_i

- edges in E

- edges in E_c

- encloses domain D_i = node of N_c

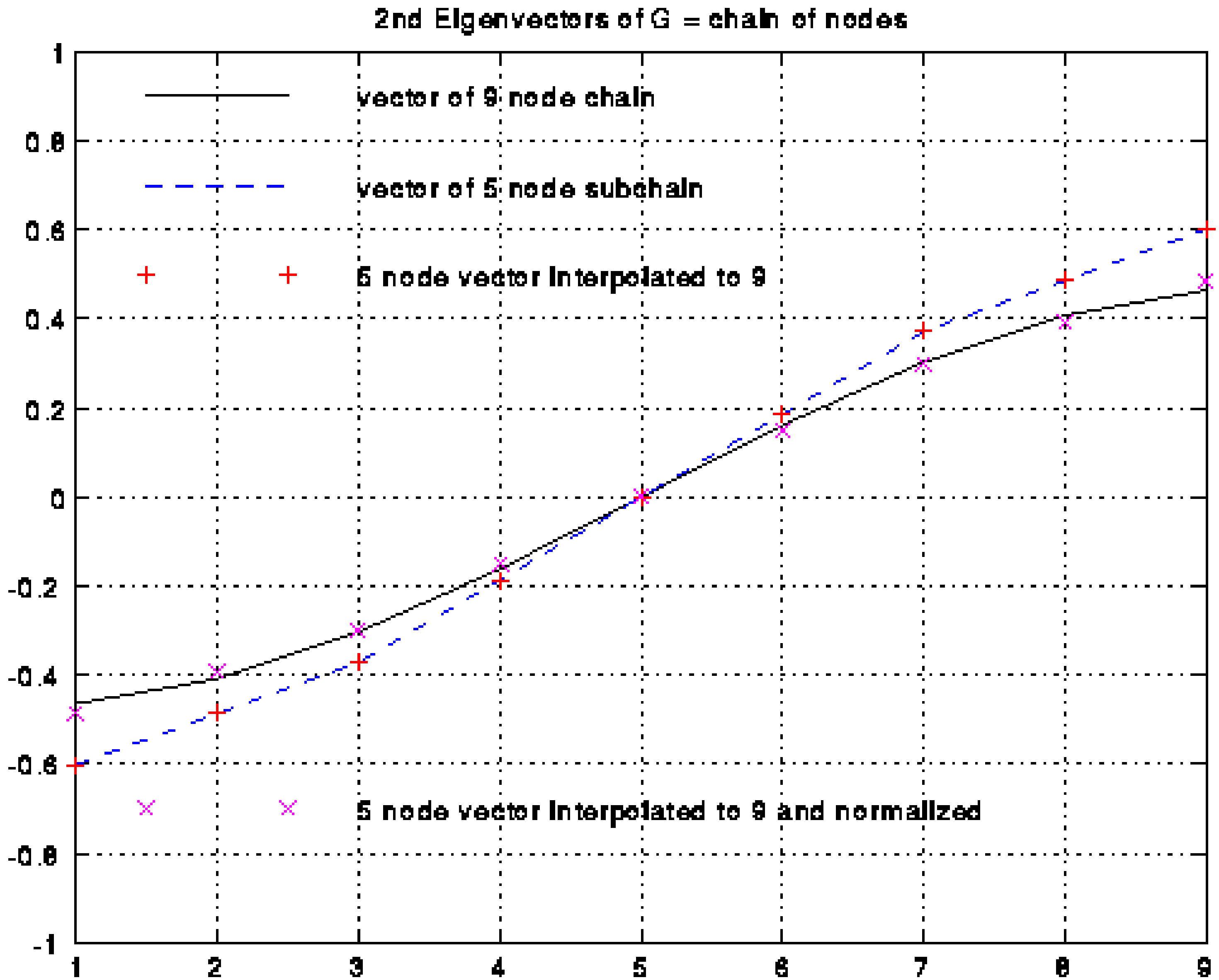
Expanding a partition of G_C to a partition of G

° Need to convert an eigenvector v_C of $L(G_C)$ to an approximate eigenvector v of L(G)

° Use interpolation:

```
For each node j in N
    if j is also a node in N_C, then
        v(j) = v_C(j)    … use same eigenvector component
    else
        v(j) = average of v_C(k) for all neighbors k of j in N_C
    end if
endif
```

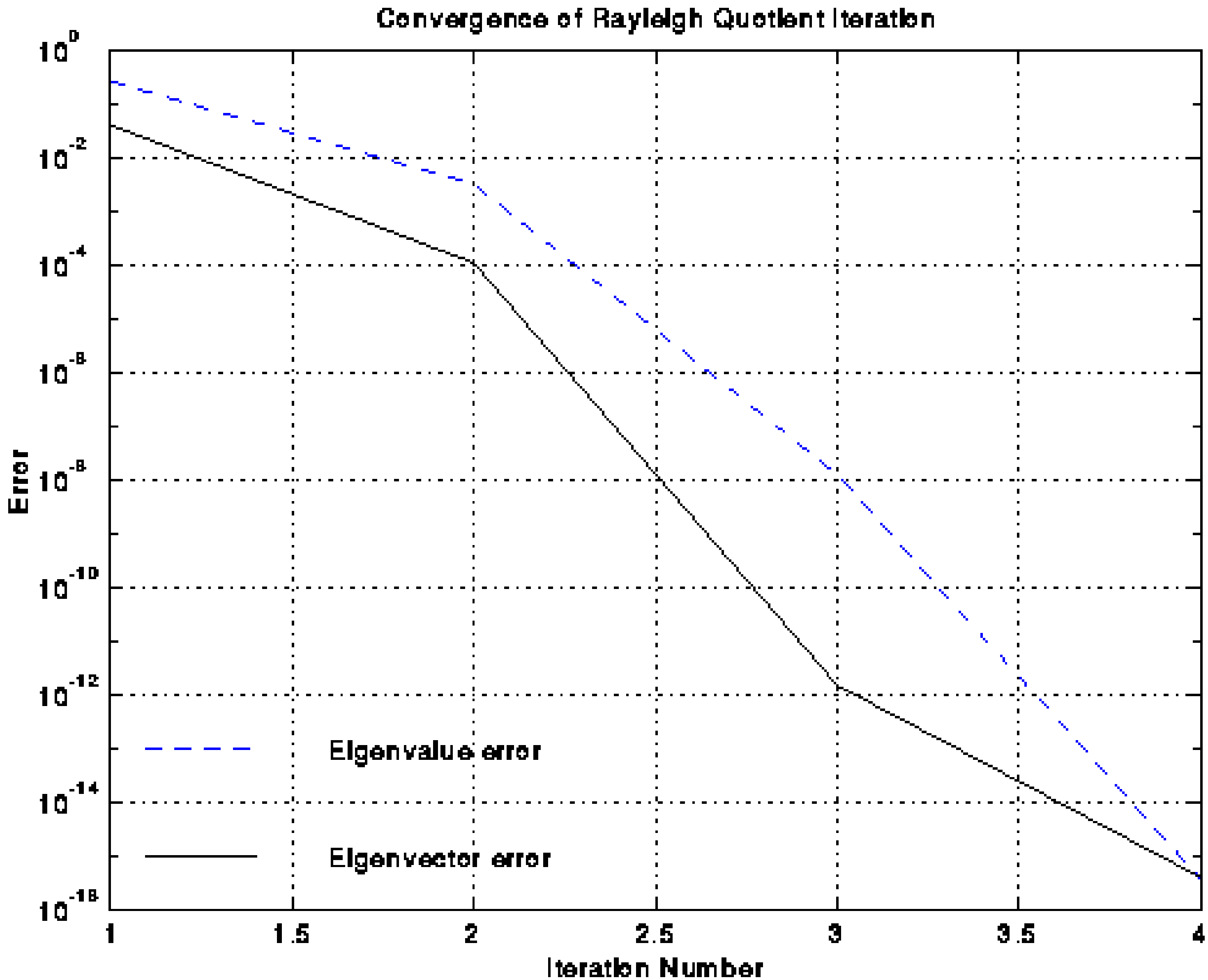
Example: 1D mesh of 9 nodes



Improve eigenvector v using Rayleigh Quotient Iteration

j = 0
pick starting vector v(0)   … from expanding v_C
repeat
    j=j+1
    $r(j) = v^T(j-1) * L(G) * v(j-1)$
    … r(j) = *Rayleigh Quotient* of v(j-1)
    …   = good approximate eigenvalue
    $v(j) = (L(G) - r(j)*I)^{-1} * v(j-1)$
    … expensive to do exactly, so solve approximately
    … using an iteration called SYMMLQ,
    … which uses matrix-vector multiply (no surprise)
    v(j) = v(j) / || v(j) || … normalize v(j)
until v(j) converges
… Convergence is very fast: cubic

Example of convergence for 1D mesh



Available Implementations

° **Multilevel Kernighan/Lin**
  - METIS (www.cs.umn.edu/~metis)
  - ParMETIS - parallel version

° **Multilevel Spectral Bisection**
  - S. Barnard and H. Simon, "A fast multilevel implementation of recursive spectral bisection …", Proc. 6th SIAM Conf. On Parallel Processing, 1993
  - Chaco (www.cs.sandia.gov/CRF/papers_chaco.html)

° **Hybrids possible**
  - Ex: Using Kernighan/Lin to improve a partition from spectral bisection

Comparison of methods

° Compare only methods that use edges, not nodal coordinates
  - CS267 webpage and KK95a (see below) have other comparisons

° Metrics
  - Speed of partitioning
  - Number of edge cuts
  - Other application dependent metrics

° Summary
  - No one method best
  - Multi-level Kernighan/Lin fastest by far, comparable to Spectral in the number of edge cuts
    - www-users.cs.umn.edu/~karypis/metis/publications/mail.html
    - see publications KK95a and KK95b
  - Spectral give much better cuts for some applications
    - Ex: image segmentation
    - www.cs.berkeley.edu/~jshi/Grouping/overview.html
    - see "Normalized Cuts and Image Segmentation"

Test matrices, and number of edges cut for a 64-way partition

For Multilevel Kernighan/Lin, as implemented in METIS (see KK95a)

Graph	# of Nodes	# of Edges	# Edges cut for 64-way partition	Expected # cuts for 2D mesh	Expected # cuts for 3D mesh	Description
144	144649	1074393	88806	6427	31805	3D FE Mesh
4ELT	15606	45878	2965	2111	7208	2D FE Mesh
ADD32	4960	9462	675	1190	3357	32 bit adder
AUTO	448695	3314611	194436	11320	67647	3D FE Mesh
BBMAT	38744	993481	55753	3326	13215	2D Stiffness M.
FINAN512	74752	261120	11388	4620	20481	Lin. Prog.
LHR10	10672	209093	58784	1746	5595	Chem. Eng.
MAP1	267241	334931	1388	8736	47887	Highway Net.
MEMPLUS	17758	54196	17894	2252	7856	Memory circuit
SHYY161	76480	152002	4365	4674	20796	Navier-Stokes
TORSO	201142	1479989	117997	7579	39623	3D FE Mesh

Expected # cuts for 64-way partition of 2D mesh of n nodes

$$n^{1/2} + 2*(n/2)^{1/2} + 4*(n/4)^{1/2} + \ldots + 32*(n/32)^{1/2} \sim 17 * n^{1/2}$$

Expected # cuts for 64-way partition of 3D mesh of n nodes =

$$n^{2/3} + 2*(n/2)^{2/3} + 4*(n/4)^{2/3} + \ldots + 32*(n/32)^{2/3} \sim 11.5 * n^{2/3}$$

Speed of 256-way partitioning (from KK95a)

Partitioning time in seconds

Graph	# of Nodes	# of Edges	Multilevel Spectral Bisection	Multilevel Kernighan/ Lin	Description
144	144649	1074393	607.3	48.1	3D FE Mesh
4ELT	15606	45878	25.0	3.1	2D FE Mesh
ADD32	4960	9462	18.7	1.6	32 bit adder
AUTO	448695	3314611	2214.2	179.2	3D FE Mesh
BBMAT	38744	993481	474.2	25.5	2D Stiffness M.
FINAN512	74752	261120	311.0	18.0	Lin. Prog.
LHR10	10672	209093	142.6	8.1	Chem. Eng.
MAP1	267241	334931	850.2	44.8	Highway Net.
MEMPLUS	17758	54196	117.9	4.3	Memory circuit
SHYY161	76480	152002	130.0	10.1	Navier-Stokes
TORSO	201142	1479989	1053.4	63.9	3D FE Mesh

Kernighan/Lin much faster than Spectral Bisection!